

# P99 CONF

## Overcoming Variable Payloads to Optimize for Performance



**Armin Ronacher**

Principal Architect at Sentry

# Armin Ronacher

Principal Architect at Sentry

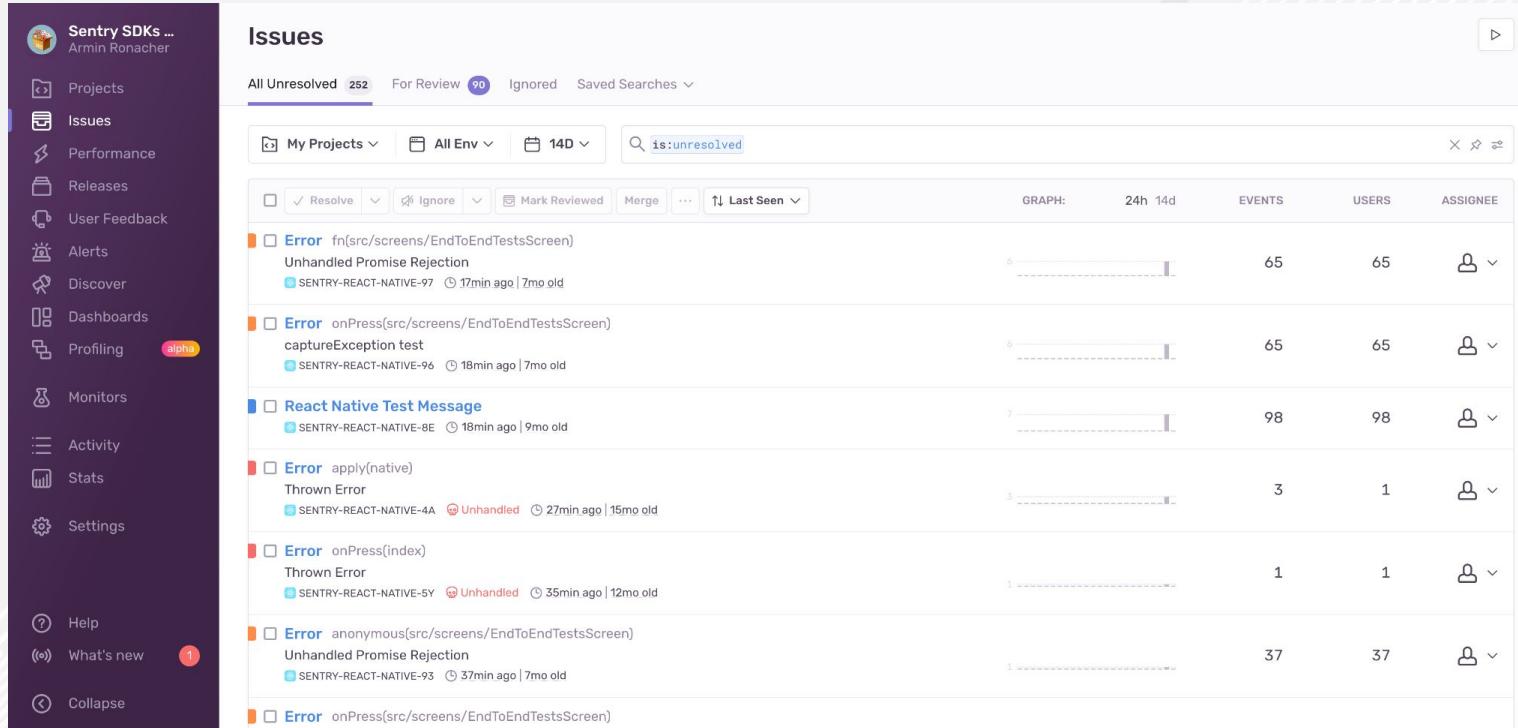


- Creator of Flask, Werkzeug, Jinja and many Open Source libs
- Keep things running at Sentry, make event processing go vroom
- Got to learn to love event processing pipelines
- Juggling three lovely kids



# Why Are We Here?

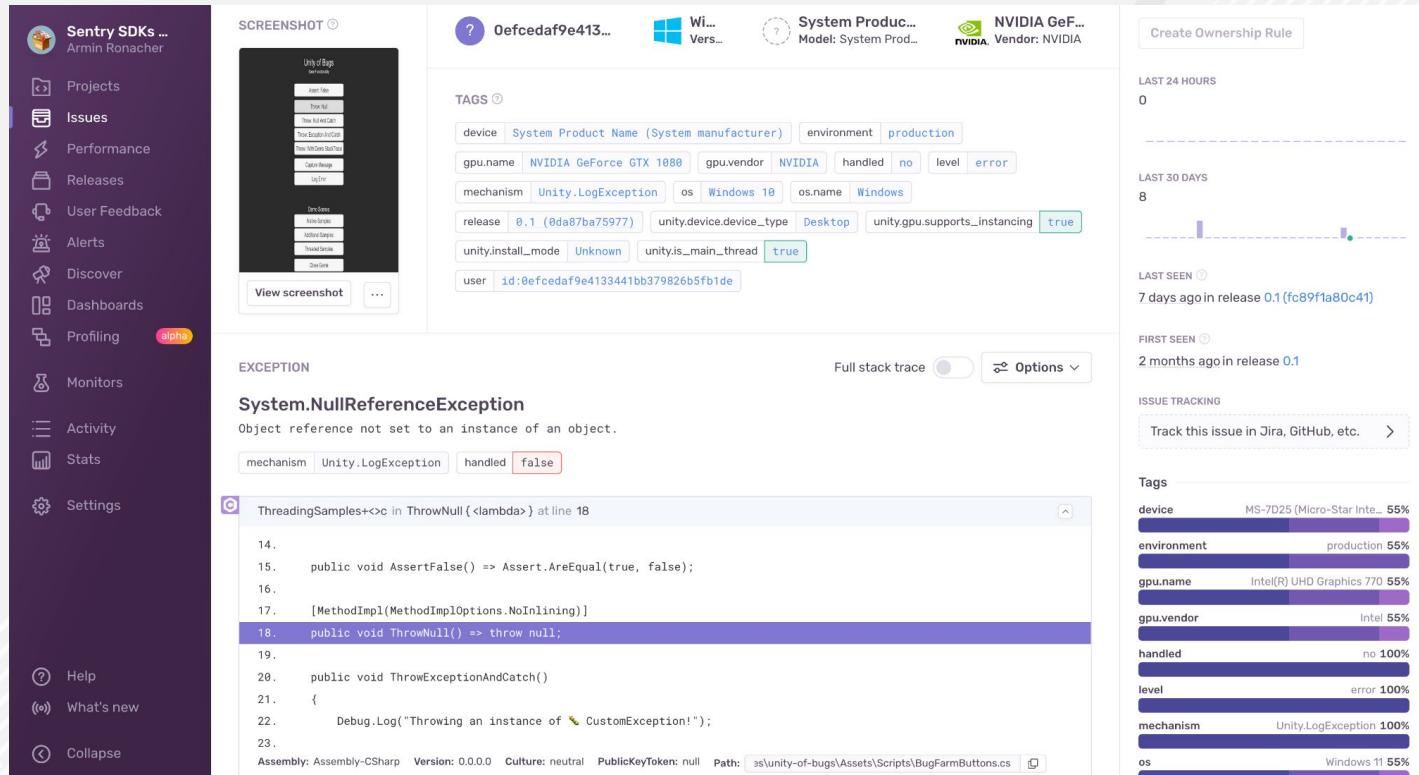
# Sentry Generates, Processes and Shows Events



The screenshot shows the Sentry interface for managing issues. The left sidebar has a dark purple theme with white text and icons. It includes links for Sentry SDKs, Armin Ronacher, Projects, Issues (which is the active tab), Performance, Releases, User Feedback, Alerts, Discover, Dashboards, Profiling (with an alpha badge), Monitors, Activity, Stats, Settings, Help, What's new (with a red badge), and Collapse. The main content area is titled 'Issues' and shows a list of 'All Unresolved' issues (252). The search bar includes filters for 'My Projects', 'All Env', '14D', and a search term 'is:unresolved'. Below the search bar are buttons for 'Resolve', 'Ignore', 'Mark Reviewed', 'Merge', and 'Last Seen'. The table lists several error types with their descriptions, timestamps, and Sentry IDs. Each row includes a checkbox, a severity icon, the error type, the function name, the error message, the timestamp, and the Sentry ID. The table also includes columns for 'GRAPH', '24h', '14d', 'EVENTS', 'USERS', and 'ASSIGNEE'.

	GRAPH:	24h	14d	EVENTS	USERS	ASSIGNEE
<input type="checkbox"/> <span style="color: orange;">Error</span> fn(src/screens/EndToEndTestsScreen)	6			65	65	
Unhandled Promise Rejection						
SENTRY-REACT-NATIVE-97	17min ago   7mo old					
<input type="checkbox"/> <span style="color: orange;">Error</span> onPress(src/screens/EndToEndTestsScreen)	6			65	65	
captureException test						
SENTRY-REACT-NATIVE-96	18min ago   7mo old					
<input type="checkbox"/> <span style="color: blue;">React Native Test Message</span>	7			98	98	
SENTRY-REACT-NATIVE-8E	18min ago   9mo old					
<input type="checkbox"/> <span style="color: red;">Error</span> apply(native)	3			1	1	
Thrown Error						
SENTRY-REACT-NATIVE-4A	Unhandled 27min ago   15mo old					
<input type="checkbox"/> <span style="color: red;">Error</span> onPress(index)	1			1	1	
Thrown Error						
SENTRY-REACT-NATIVE-5Y	Unhandled 35min ago   12mo old					
<input type="checkbox"/> <span style="color: orange;">Error</span> anonymous(src/screens/EndToEndTestsScreen)	37			37	37	
Unhandled Promise Rejection						
SENTRY-REACT-NATIVE-93	37min ago   7mo old					
<input type="checkbox"/> <span style="color: orange;">Error</span> onPress(src/screens/EndToEndTestsScreen)						

# Sentry Generates, Processes and Shows Events



The screenshot displays the Sentry interface for managing and analyzing application events. The left sidebar contains navigation links for Sentry SDKs, Projects, Issues, Performance, Releases, User Feedback, Alerts, Discover, Dashboards, Profiling (marked as alpha), Monitors, Activity, Stats, and Settings. The main content area is divided into several sections:

- SCREENSHOT**: Shows a screenshot of a Unity application interface titled "Unity of Bugs". The screenshot displays a list of bugs with columns for "Issue ID", "Title", "Type", "Priority", "Status", and "Last Seen".
- Event Details**: Shows event details for a specific event ID: `Defcedaf9e413...`. It includes system information (Windows 10, System Product Name: System Product Model, GPU: NVIDIA GeForce GTX 1080, Vendor: NVIDIA), tags, and a stack trace for a `System.NullReferenceException`.
- Stack Trace**: Displays the C# code for the `ThrowNull` method, showing the line of code that caused the exception.
- Timeline**: A timeline showing event counts for the last 24 hours (0), last 30 days (8), and last seen (7 days ago in release 0.1).
- Issue Tracking**: A button to "Track this issue in Jira, GitHub, etc."
- Tags**: A chart showing the distribution of various tags across the events. The most common tags are device (MS-7D25), environment (production), gpu.name (Intel(R) UHD Graphics 770), gpu.vendor (Intel), handled (no), level (error), mechanism (Unity.LogException), and os (Windows 11).

# Sentry Events

- Session Updates
- Transaction Events
- Metrics
- Reports
  - Messages
  - Structured Processed Crash Reports
  - Structured Unprocessed Crash Reports
  - Minidumps
  - Third Party Crash Formats
  - User Feedback
  - Profiles
  - Attachments
  - Client Reports

# Challenges

- Users want crash reports with low latency
- Variance of processing times of events from 1ms to 30 minutes
- How long an event takes, is not always known ahead of time
- What happens at the end of the pipeline can affect the beginning of it
- Part of the pipeline is an Onion that can extend closer and closer to the user

# Conservative Changes

# Touching Running Systems

- Sentry processes complex events from many sources
- Any change (even bugfix) can break someone's workflow
- We are treating very carefully

Things we try to avoid doing:

- Bumping Dependencies without reason
- Rewriting services as busywork

That doesn't mean we don't change the pipeline, but we are rather conservative.

# Terms and Things

# “The Monolith”

- Written in Python
- A massive and grown Django app
- Uses celery and rabbitmq historically for all queue needs
- Still plays a significant role in the processing logic
- Uses CFFI to invoke some Rust code

# Relay

- Written in Rust
- Our ingestion component
- Layers like an onion
- Stateful
- First level quota enforcement
- Aggregation
- Data normalization
- PII stripping

# Symbolicator

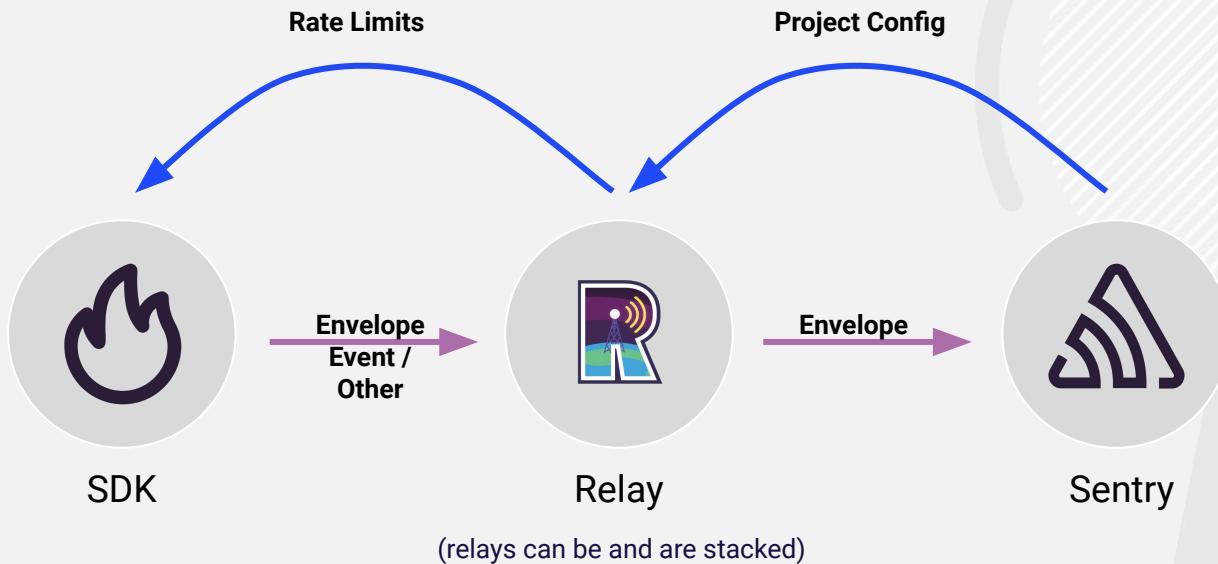
- Written in Rust
- Handles Symbolication
  - PDB
  - PE/COFF
  - DWARF
  - MachO
  - ELF
  - WASM
  - IL2CPP
- Fetches and Manages Debug Information Files (DIFs)
  - External Symbol Servers
  - Internal Sources

# Ingest Consumer

- Shovels Pieces from the Relay supplied Kafka stream onwards
  - Events
  - User Reports
  - Attachment Chunks
  - Attachments
- Does an initial routing of events to the rest of pipeline

# What's Flowing?

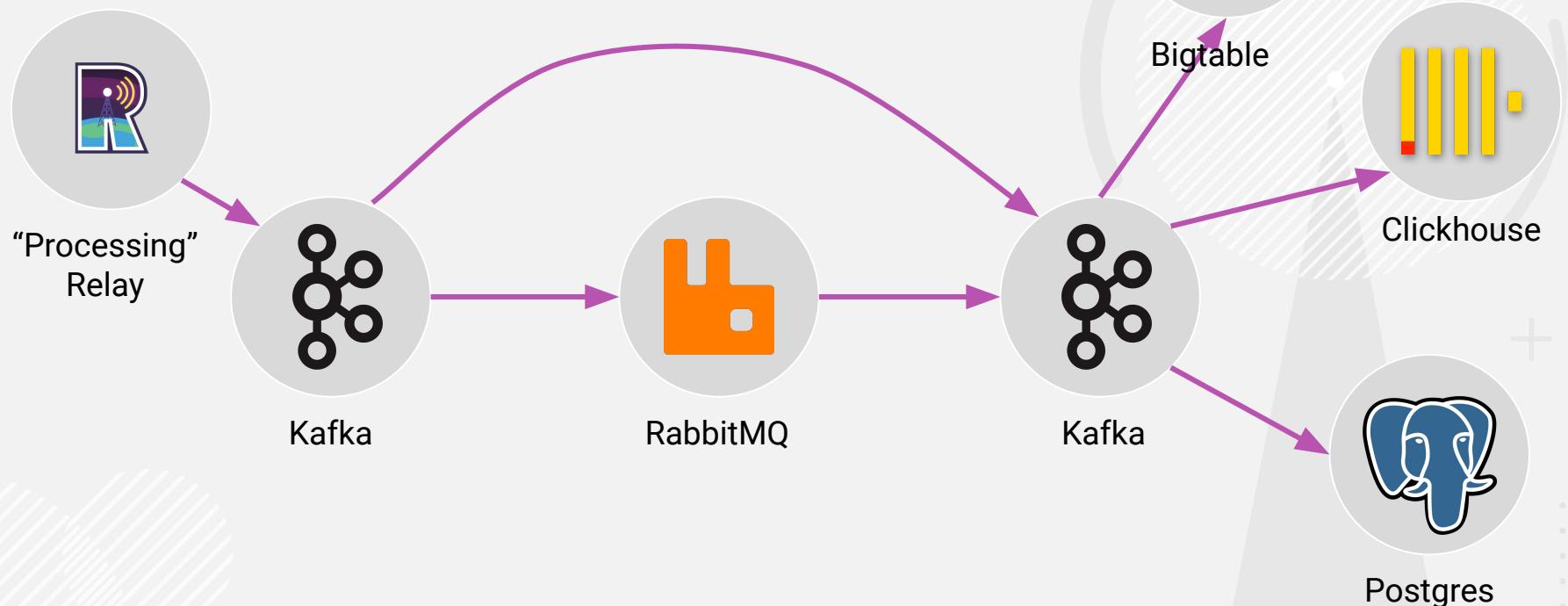
# Ingestion Side



# Ingestion Traffic

- POP Relays accepts around 100k events/sec at regular day peak and rejects around 40k/sec
- Processing relays process around 150k events/sec at regular day peak
- Global Ingestion-Level Load Balancers see around 200k req/sec at regular peak

# Processing Side



# Kafka Traffic

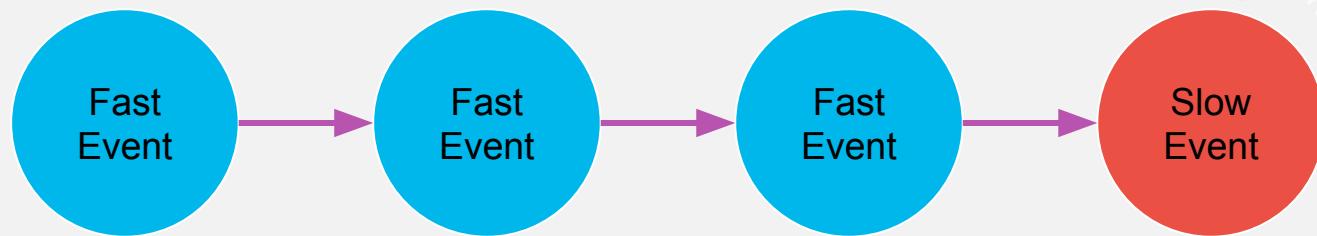
- All relay traffic makes it to different Kafka topics
- Important ones by volume:
  - Sessions/Metrics
  - Transactions
  - Error events
  - Attachments
- Based on these event types, initial routing happens
- **The biggest challenge are error events**

# Error Event Routing

- Ahead of time, little information is available to determine how long an event will take
- Cache status can greatly affect how long it takes
  - JavaScript event without source maps can take <1ms
  - JavaScript event that requires fetching of source maps can take 60sec or more
  - Native events might pull in gigabytes of debug data, that's not yet hot
- A lot of that processing still happens in legacy monolith

# The Issue with Variance

# Head of Line Blocking within Partition



# Our Queues: Kafka and RabbitMQ

- Kafka has inherent head-of-line blocking
- Our Python consumers have language limited support for concurrency
- Writing a custom broker on top of Kafka carries risks
- Historically our answer was to dispatch from Kafka to Rabbit for high variance tasks

# We're Not Happy with RabbitMQ

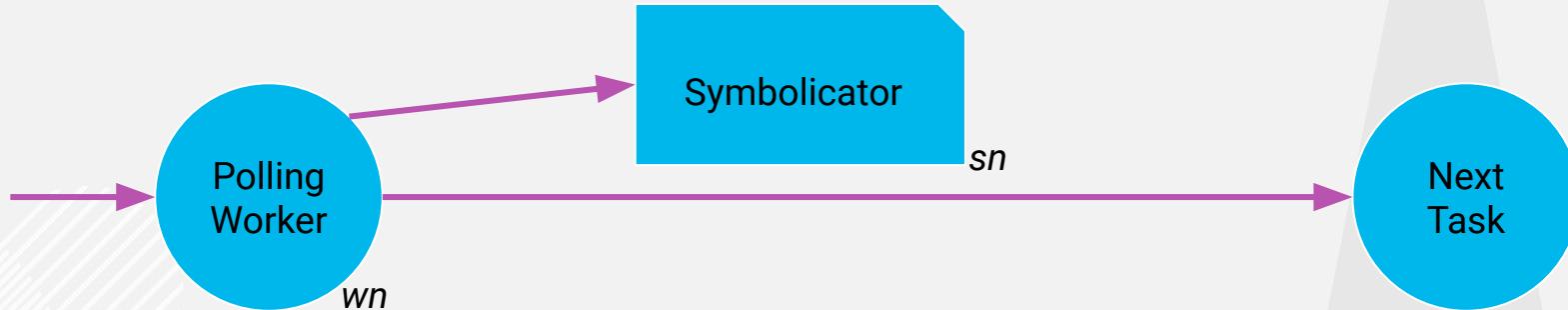
- As our scale increases, we likely will move to Kafka entirely
- This switch will require us to build a custom broker
- So far the benefits of that have not yet emerged
- It works good enough for now™

# Tasks on RabbitMQ

- Tasks travel on RabbitMQ queues
- Event payloads live in redis
- Python workers pick up tasks as they have capacity available
- Problem: polling workers

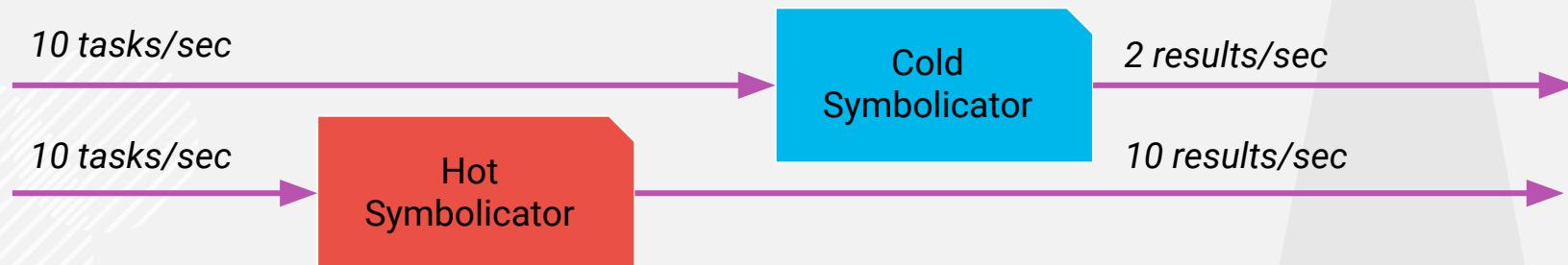
# Polling Workers

- Some tasks poll the internal symbolicator service
- For that a Python worker dispatches a task via HTTP to the stateful symbolicator service
- Python worker polls that service until result is ready which can be minutes
- Requires symbolicators to be somewhat evenly configured and loaded



# Incident: Symbolicator Tilt

- Fundamental flaw: tasks are pushed evenly to symbolicators
- Not all symbolicators respond the same
- A freshly scaled up symbolicator has cold caches
- This caused scaling up to have a negative effect on processing times
- Workaround: **cache sharing**
- Long term plan: symbolicator picks up directly from RabbitMQ or Kafka



# Backpressure Control

# Implicit Backpressure Control

- Our processing queue has insufficient backpressure control
- At the head of the queue we permit almost unbounded event accumulation
- Pausing certain parts of the pipeline can cause it to spill too fast into RabbitMQ (goes to swap)

# Deep Load Shedding

# Pipeline Kill-Switches

- Problem: for some reason bad event data makes it into the pipeline
- Due to volume we cannot track where the data is in the pipe and we likely can't reliably prevent it from propagating further
- Solution: flexible kill-switches
- Drop events that match a filter wherever that filter is applied

# Loading Kill-Switches

```
sentry killswitches pull \  
  store.load-shed-group-creation-projects \  
  new-rules.txt
```

Before: <disabled entirely>

After:

```
DROP DATA WHERE  
  (project_id = 1) OR  
  (project_id = 2) OR  
  (project_id = 3)
```

Should the changes be applied? [y/N]: y

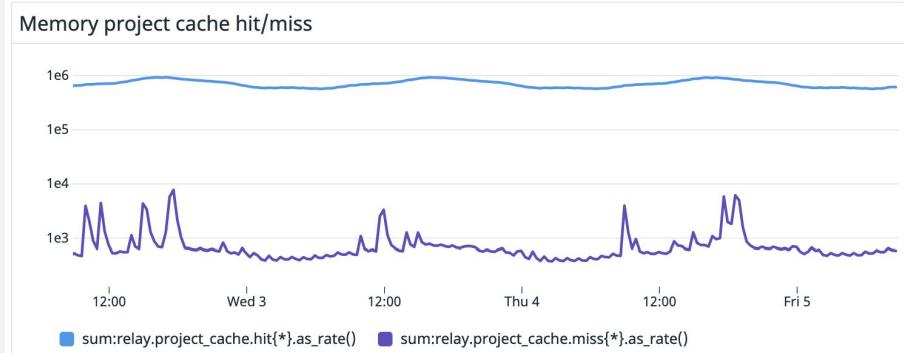
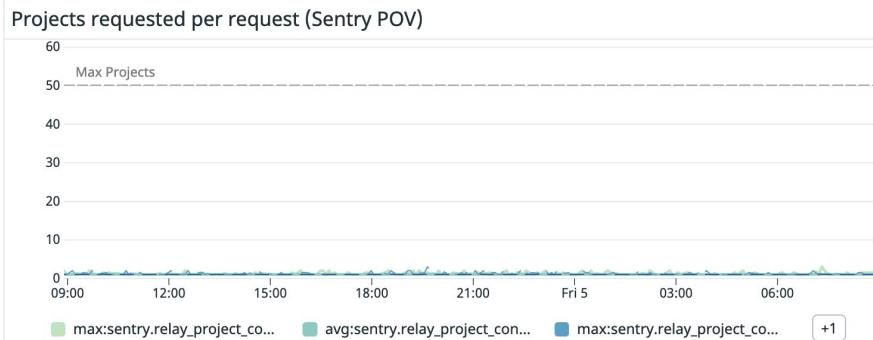
# Look into Relay

# Communication Channels

- Relay to Relay: HTTP
- Relay to Processing Pipeline: Kafka
- Relay state updates:
  - Relay -> Relay via HTTP
  - Relay to Internal HTTP and direct redis cache reads

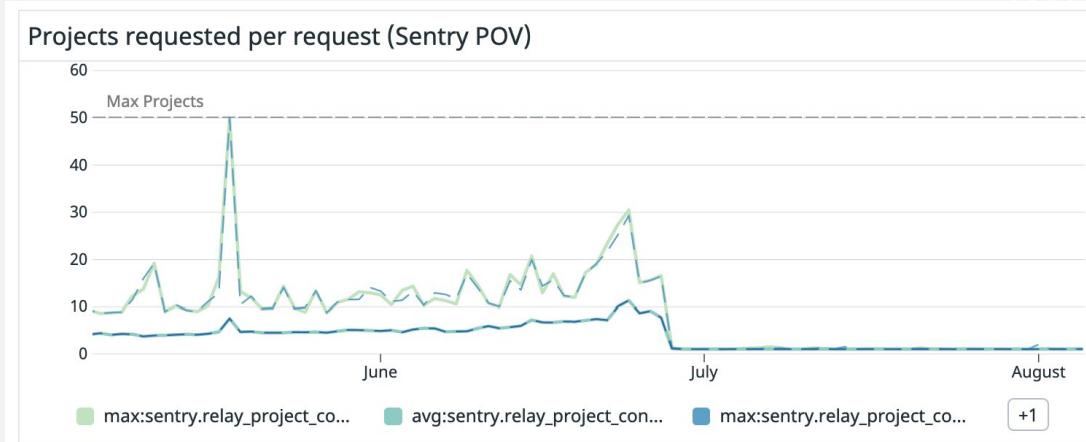
# Project Config Caches

- Innermost relays fetch config directly from Sentry
- Sentry itself persists latest config into redis
- Relay will always try to read from that shared cache before asking Sentry



# Proactive Cache Writing

- We used to expire configs in cache liberally
- Now most situations will instead proactively rewrite configs to cache





Armin Ronacher  
[armin@sentry.io](mailto:armin@sentry.io)  
@mitsuhiko